/\*

u8g\_dev\_ssd1322\_nhd31oled\_gr.c

2-Bit (4L) Driver for SSD1322 Controller (OLED Display)

Tested with NHD-3.12-25664

Universal 8bit Graphics Library

Copyright (c) 2012, olikraus@gmail.com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,

are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this list

of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice, this

list of conditions and the following disclaimer in the documentation and/or other

materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND

CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,

INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT

NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,

STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF

ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

SSD130x Monochrom OLED Controller

SSD131x Character OLED Controller

SSD132x Graylevel OLED Controller

SSD1331 Color OLED Controller

\*/

#include "u8g.h"

/\* width must be multiple of 8, largest value is 248 unless u8g 16 bit mode is enabled \*/

#if defined(U8G\_16BIT)

#define WIDTH 256

#else

#define WIDTH 248

#endif

#define HEIGHT 64

//#define PAGE\_HEIGHT 8

/\*

http://www.newhavendisplay.com/app\_notes/OLED\_25664.txt

http://www.newhavendisplay.com/forum/viewtopic.php?f=15&t=3758

\*/

static const uint8\_t u8g\_dev\_ssd1322\_2bit\_nhd\_312\_init\_seq[] PROGMEM = {

U8G\_ESC\_DLY(10), /\* delay 10 ms \*/

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_RST(1), /\* do reset low pulse with (1\*16)+2 milliseconds \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

U8G\_ESC\_DLY(100), /\* delay 100 ms \*/

U8G\_ESC\_DLY(100), /\* delay 100 ms \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0fd, /\* lock command \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x012, /\* unlock \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0ae, /\* display off, sleep mode \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0b3,

U8G\_ESC\_ADR(1), /\* data mode \*/

0x091, /\* set display clock divide ratio/oscillator frequency (set clock as 80 frames/sec) \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0ca, /\* multiplex ratio \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x03f, /\* 1/64 Duty (0x0F~0x3F) \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0a2,

U8G\_ESC\_ADR(1), /\* data mode \*/

0x000, /\* display offset, shift mapping ram counter \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0a1,

U8G\_ESC\_ADR(1), /\* data mode \*/

0x000, /\* display start line \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0a0, /\* Set Re-Map / Dual COM Line Mode \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x014, /\* was 0x014 \*/

0x011, /\* was 0x011 \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0ab,

U8G\_ESC\_ADR(1), /\* data mode \*/

0x001, /\* Enable Internal VDD Regulator \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0b4, /\* Display Enhancement A \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x0a0,

0x005|0x0fd,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0c1, /\* contrast \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x09f,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0c7, /\* Set Scale Factor of Segment Output Current Control \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x00f,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0b9, /\* linear gray scale \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0b1, /\* Phase 1 (Reset) & Phase 2 (Pre-Charge) Period Adjustment \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x0e2,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0d1, /\* Display Enhancement B \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x082|0x020,

0x020,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0bb, /\* precharge voltage \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x01f,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0b6, /\* precharge period \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x008,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0be, /\* vcomh \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x007,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0a6, /\* normal display \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0a9, /\* exit partial display \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x0af, /\* display on \*/

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_END /\* end of sequence \*/

};

static const uint8\_t u8g\_dev\_ssd1322\_2bit\_nhd\_312\_prepare\_page\_seq[] PROGMEM = {

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

0x015, /\* column address... \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

0x01c, /\* start at column 0 \*/

0x05b, /\* end column \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x075, /\* row address... \*/

U8G\_ESC\_ADR(1), /\* data mode \*/

U8G\_ESC\_END /\* end of sequence \*/

};

static void u8g\_dev\_ssd1322\_2bit\_prepare\_row(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t delta\_row)

{

uint8\_t row = ((u8g\_pb\_t \*)(dev->dev\_mem))->p.page;

row \*= ((u8g\_pb\_t \*)(dev->dev\_mem))->p.page\_height;

row += delta\_row;

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1322\_2bit\_nhd\_312\_prepare\_page\_seq);

u8g\_WriteByte(u8g, dev, row); /\* start at the selected row \*/

u8g\_WriteByte(u8g, dev, row+1); /\* end within the selected row \*/

u8g\_SetAddress(u8g, dev, 0); /\* instruction mode mode \*/

u8g\_WriteByte(u8g, dev, 0x05c); /\* write to ram \*/

u8g\_SetAddress(u8g, dev, 1); /\* data mode \*/

}

static const uint8\_t u8g\_dev\_ssd13xx\_sleep\_on[] PROGMEM = {

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

0x0ae, /\* display off \*/

U8G\_ESC\_CS(0), /\* disable chip, bugfix 12 nov 2014 \*/

U8G\_ESC\_END /\* end of sequence \*/

};

static const uint8\_t u8g\_dev\_ssd13xx\_sleep\_off[] PROGMEM = {

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

0x0af, /\* display on \*/

U8G\_ESC\_DLY(50), /\* delay 50 ms \*/

U8G\_ESC\_CS(0), /\* disable chip, bugfix 12 nov 2014 \*/

U8G\_ESC\_END /\* end of sequence \*/

};

uint8\_t u8g\_dev\_ssd1322\_nhd31oled\_gr\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

switch(msg)

{

case U8G\_DEV\_MSG\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_300NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1322\_2bit\_nhd\_312\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

uint8\_t i;

u8g\_pb\_t \*pb = (u8g\_pb\_t \*)(dev->dev\_mem);

uint8\_t \*p = pb->buf;

u8g\_uint\_t cnt;

cnt = pb->width;

cnt >>= 2;

for( i = 0; i < pb->p.page\_height; i++ )

{

u8g\_dev\_ssd1322\_2bit\_prepare\_row(u8g, dev, i); /\* this will also enable chip select \*/

#if !defined(U8G\_16BIT)

u8g\_WriteByte(u8g, dev, 0x00);

u8g\_WriteByte(u8g, dev, 0x00);

#endif

u8g\_WriteSequence4LTo16GrDevice(u8g, dev, cnt, p);

#if !defined(U8G\_16BIT)

u8g\_WriteByte(u8g, dev, 0x00);

u8g\_WriteByte(u8g, dev, 0x00);

#endif

u8g\_MicroDelay(); // for DUE?

u8g\_SetChipSelect(u8g, dev, 0);

p+=cnt;

}

}

break;

case U8G\_DEV\_MSG\_CONTRAST:

u8g\_SetChipSelect(u8g, dev, 1);

u8g\_SetAddress(u8g, dev, 0); /\* instruction mode \*/

u8g\_WriteByte(u8g, dev, 0x081);

u8g\_SetAddress(u8g, dev, 1); /\* data mode \*/

u8g\_WriteByte(u8g, dev, (\*(uint8\_t \*)arg) >> 1);

u8g\_MicroDelay(); // for DUE?

u8g\_SetChipSelect(u8g, dev, 0);

break;

case U8G\_DEV\_MSG\_SLEEP\_ON:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_on);

return 1;

case U8G\_DEV\_MSG\_SLEEP\_OFF:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_off);

return 1;

}

return u8g\_dev\_pb8h2\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

switch(msg)

{

case U8G\_DEV\_MSG\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_300NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1322\_2bit\_nhd\_312\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

uint8\_t i;

u8g\_pb\_t \*pb = (u8g\_pb\_t \*)(dev->dev\_mem);

uint8\_t \*p = pb->buf;

u8g\_uint\_t cnt;

cnt = pb->width;

cnt >>= 2; /\* 23 Oct 2013, changed to 2 \*/

for( i = 0; i < pb->p.page\_height; i++ )

{

u8g\_dev\_ssd1322\_2bit\_prepare\_row(u8g, dev, i); /\* this will also enable chip select \*/

#if !defined(U8G\_16BIT)

u8g\_WriteByte(u8g, dev, 0x00);

u8g\_WriteByte(u8g, dev, 0x00);

#endif

u8g\_WriteSequence4LTo16GrDevice(u8g, dev, cnt, p);

#if !defined(U8G\_16BIT)

u8g\_WriteByte(u8g, dev, 0x00);

u8g\_WriteByte(u8g, dev, 0x00);

#endif

u8g\_MicroDelay(); // for DUE?

u8g\_SetChipSelect(u8g, dev, 0);

p+=cnt;

}

}

break;

case U8G\_DEV\_MSG\_CONTRAST:

u8g\_SetChipSelect(u8g, dev, 1);

u8g\_SetAddress(u8g, dev, 0); /\* instruction mode \*/

u8g\_WriteByte(u8g, dev, 0x081);

u8g\_SetAddress(u8g, dev, 1); /\* data mode \*/

u8g\_WriteByte(u8g, dev, (\*(uint8\_t \*)arg) >> 1);

u8g\_SetChipSelect(u8g, dev, 0);

break;

case U8G\_DEV\_MSG\_SLEEP\_ON:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_on);

return 1;

case U8G\_DEV\_MSG\_SLEEP\_OFF:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_off);

return 1;

}

return u8g\_dev\_pb16h2\_base\_fn(u8g, dev, msg, arg);

}

U8G\_PB\_DEV(u8g\_dev\_ssd1322\_nhd31oled\_gr\_sw\_spi , WIDTH, HEIGHT, 4, u8g\_dev\_ssd1322\_nhd31oled\_gr\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1322\_nhd31oled\_gr\_hw\_spi , WIDTH, HEIGHT, 4, u8g\_dev\_ssd1322\_nhd31oled\_gr\_fn, U8G\_COM\_HW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1322\_nhd31oled\_gr\_parallel , WIDTH, HEIGHT, 4, u8g\_dev\_ssd1322\_nhd31oled\_gr\_fn, U8G\_COM\_FAST\_PARALLEL);

#define DWIDTH (WIDTH\*2)

uint8\_t u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_buf[DWIDTH] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_pb = { {8, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_buf};

u8g\_dev\_t u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_sw\_spi = { u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_fn, &u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_hw\_spi = { u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_fn, &u8g\_dev\_ssd1322\_nhd31oled\_2x\_gr\_pb, U8G\_COM\_HW\_SPI };